# Single failure resiliency in greedy routing

Sahel Sahhaf*, Wouter Tavernier*, Didier Colle*, Mario Pickavet* and Piet Demeester*
*Department of Information Technology (INTEC), Ghent University, iMinds
Gaston Crommenlaan 8, 9050 Gent, Belgium
Email:{sahel.sahhaf, wouter.tavernier, didier.colle, mario.pickavet, piet.demeester}@intec.ugent.be

***Abstract*—Using greedy routing, network nodes forward packets towards neighbors which are closer to their destination. This approach makes greedy routers significantly more memory-efficient than traditional IP-routers using longest-prefix matching. Greedy embeddings map network nodes to coordinates, such that greedy routing always leads to the destination. Prior works showed that using a spanning tree of the network topology, greedy embeddings can be found in different metric spaces for any graph. However, a single link/node failure might affect the greedy embedding and causes the packets to reach a dead end. In order to cope with network failures, existing greedy methods require large resources and cause significant loss in the quality of the routing (stretch loss). We propose efficient recovery techniques which require very limited resources with minor effect on the stretch. As the proposed techniques are protection, the switch-over takes place very fast. Low overhead, simplicity and scalability of the methods make them suitable for large-scale networks. The proposed schemes are validated on large topologies with properties similar to the Internet. The performances of the schemes are compared with an existing alternative referred as gravity pressure routing.**

## I. Introduction

Next generation Internet protocols are required to increase performance, capacity and packet forwarding rates in order to meet future demands. Not only throughput requirements are increasing but also the size of the routing tables are growing ([1] reports more than 400K RIB entries in current BGP routers) and in near future they are expected to increase to 2 million entries. As traditional IP routing is based on longest prefix matching in a routing table, the scalability of current routing protocols is limited by the size of the corresponding routing tables. Greedy routing can be considered as an alternative to IP routing for future Internet. This routing scheme does not require address lookup in order to find the next hop and it is more memory efficient than IP routing based on longest prefix matching.

In greedy routing, every node in the network is assigned a coordinate in a metric space. Based on these coordinates, every node forwards the incoming packets to a neighbor which is closer to the destination of the packets. Using this scheme, nodes are only required to store the coordinates of their neighbors in order to make routing decisions. Network coordinates can be based on physical locations of the nodes [2], leading to the term *geographic routing* or virtual coordinates embedded in a geometric space different from the physical one, *geometric routing* [3]. A problem of these schemes is that packets might reach a local minimum (lake or void), meaning that there is no neighbor which is closer to the packet's intended destination than the current node. Greedy embeddings are proposed as a solution to this issue [4]. In a formal way, a greedy embedding for a given graph $G(V, E)$ into a metric space $X$, is a function from $V(G)$ to $X$ such that for all graph nodes $s \neq t$, $s$ has a neighbor $u$ which decreases the distance toward $t$ in metric space $X$. Greedy embeddings guarantee 100% successful delivery to every destination in the network.

Robert Kleinberg proved that coordinates for nodes of any type of graph can be found in a two-dimensional hyperbolic space such that greedy routing is always successful [5]. The derivation of the coordinates is based on a spanning tree of the network graph. However, Kleinberg did not investigate the effect of dynamics in the network topology. We propose another greedy embedding which is also based on the spanning tree of the network. However, we use a very simple numbering of the nodes and the embedding is not into a special metric space. The proposed greedy embedding guarantees a distance-decreasing path via the spanning tree of the network. Therefore, a single change in the connectivity of this tree (link/node failure) might affect the greedy embedding. This might lead the packets to a void. Many existing approaches use face routing to bypass the void and reach a node which greedy routing can be resumed from [6]. Unfortunately, face routing techniques suffer of two types of issues: i) local graph parts must be planar (or being planarized.), and ii) the latter cannot always be guaranteed to be found. Another possible solution is the re-construction of the tree and re-calculation of the coordinates. Using this approach, it is possible that too many nodes require changing their coordinates and this might result into long disruptions.

In this paper, we propose recovery techniques to cope with network failures in greedy routing without the re-calculation of the coordinates. The proposed schemes guarantee single failure resiliency. As the methods are protection, a very fast switch-over is possible. The memory overheads both in network nodes and in the packets are very limited and the loss in the quality of the routing is insignificant compared to the existing methods. Different parts of the routing, even upon facing a failure are based on greedy routing. The methods work in such a way that the local minima are avoided in the first place. Therefore, unlike some available techniques [7], there is no need to keep track of the visited nodes along a path. Low overhead, simplicity and scalability of the methods make them suitable for distributed implementation and therefore, usage in large-scale networks.

The paper is structured as follows. In Section II, the related works for greedy routing and recovery techniques are described. Section III explains the proposed greedy embedding. The two proposed recovery methods for link and node failure are described in Section IV. Section V contains the evaluation results for the proposed methods. Future works are discussed in Section VI and finally Section VII concludes the paper.

## II. Existing work

As was mentioned earlier, face routing is one solution to cope with local minima in greedy routing. This technique tries to escape from a void, by routing around the area of the face in a planar subgraph of the network, until greedy routing can be resumed. The graph being planar is the major limitation of this technique. Using greedy embeddings is the other solution and Kleinberg proved that it is possible to find such an embedding for any graph in the hyperbolic plane [5].

Regarding the network dynamics in greedy routing, there are some methods which cope with the failures using face routing [6]. However, not so many works investigate the effect of failures in greedy routing based on greedy embeddings. In [7], the authors propose an embedding in the hyperbolic plane which allows incremental embedding to cope with node addition in the network. In order to handle node/link failures, they use an adapted routing algorithm in case a packet reaches a dead end. The routing is called Gravity-Pressure. The packets are forwarded via neighbors which decrease the distance towards the destination as much as possible. In order to avoid loops, a path trace needs to be maintained in every packet from the moment a dead end is reached. This approach causes a large overhead to the header of the packets. In [8], we proposed recovery methods for link failures. The greedy routing was based on the embedding in [5]. In this paper, we propose a very simple greedy embedding and a new recovery technique for single link failure. In the new technique, we benefit from backup trees while the technique in [8] was based on finding alternate paths for every link in the primary tree. The new technique has some advantages on the computation/communication cost compared to the technique in [8]. We also propose a recovery technique for single node failure using disjoint backup paths.

Redundant trees are commonly used for protection and restoration in communication networks. In many cases, link-disjoint or node-disjoint trees are generated to cope with network failures. As finding two completely disjoint trees is not possible in every network, maximally disjoint trees and colored trees have been proposed [9], [10]. In the latter, the methods try to find two trees rooted at the same node and every node in the network has two disjoint paths to the root in the two trees. It is proved by Itai and Rodeh [11] that for every 2-connected graph, such trees can be generated. There are some methods using multiple spanning trees to recover from a single node/link failure. In these methods, trees are found in such a way that for every possible single failure there is at least one tree which does not contain that failed element [12]. As these types of methods imply large overhead in every node, we focus on the colored trees. We use this type of trees for single link failure resiliency in greedy routing. As only two trees are required, the memory overhead in every node and also in packets is very limited. Both proposed recovery techniques are protection schemes.

## III. Greedy embedding based on the spanning tree

As explained earlier, local minima are a major drawback of greedy routing. As a solution, we propose a simple greedy embedding (numbering) of nodes which is based on the spanning tree of the network. Using the proposed embedding, every packet reaches its destination eventually.

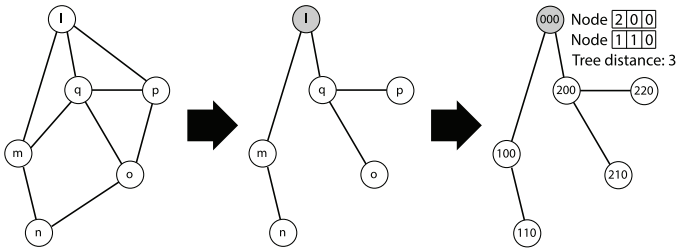


Fig. 1. Steps of network nodes numbering

### A. Nodes numbering

The general idea is to know the location of a node in the spanning tree of the network. This information is useful in order to find the distance between two nodes on the tree. This distance is used to perform the greedy routing. Any type of node numbering which indicates the location of a node in the spanning tree of the network can be used in this scheme. In the following, the required steps of a sample node numbering are explained.

1) First the spanning tree of the network is constructed.
2) The neighbors of a node in the spanning tree of the network are numbered from 1 to d (with d being the degree of the node in the tree).
3) Each node can calculate the coordinates of its children by adding the integer number assigned to each child after the last non-zero field in its own coordinate.

Note that all fields of the root coordinate are zero. The explained numbering of network nodes can be performed at the same time as spanning tree construction. The exchanged messages between nodes for tree generation can contain the coordinate of the nodes as well. Figure 1 depicts an example for the explained tree numbering (coordinate) and tree distance. The number of non-zero fields in a coordinate represents the level of that node in the spanning tree of the network. Using these coordinates, the locations of nodes in the tree can be determined easily. In this scheme, in order to determine which neighbor is closer to the destination of a packet the tree distance is calculated. The tree distance between two nodes is actually the hop count on the tree between them. In Figure 1, the tree distance between node 110 and node 200 is 3.

### B. Tree-distance calculation

In this section, the calculation of tree distance between two nodes based on the assigned tree coordinates is explained. The required steps for this calculation are as follows:

1) The closest common parent to both nodes should be found.
2) The hop counts of each node to the common parent should be calculated.
3) The sum of these two hop counts determines the tree distance.

The first common fields in the coordinates of the two nodes indicate the closest common parent to both nodes. In each coordinate, the number of non-zero fields after the first uncommon field represents the hop counts to the common

parent. The sum of hop counts of both coordinates determines the tree distance between the two nodes. In Figure 1, consider node 200 and node 110. The closest common parent for these two nodes is the root. The counting starts from the first uncommon field in both coordinates. In node 110 there are 2 nonzero fields and node 200 has only one non-zero field. Therefore, the sum of these two hop counts determines the tree distance between these two nodes which is 3.

### C. Tree-based greedy routing

Upon arrival of a packet in a node, the tree distance between every neighbor and the destination is calculated and the neighbor which decreases the distance most is selected as the next hop of the packet. This routing is different from tree routing, because the shortcuts (the edges which are not in the spanning tree) can also be taken and the tree is only used for coordinate assignment and to guarantee a distance decreasing path between every two nodes.

## IV. Recovery Techniques in Greedy Routing

In order to perform greedy routing in the network, we use the tree-numbering which was explained in Section III. This numbering is based on the spanning tree of the network which means that the spanning tree guarantees the distance-decreasing path between every two nodes of the network. Therefore, a change in the connectivity of this spanning tree might affect the greedy embedding causing the packets to reach a dead end. Figure 2 illustrates an example for a tree-edge failure and the problem of local minimum. In the left figure, the greedy path from S to D is determined by the solid line. The depicted tree-edge failure makes the node 000 a local minimum as it has no neighbor closer to D than itself. In this section, we first explain a recovery method for a link failure and then a method to cope with a node failure. Both methods are protection schemes. In the methods, we assume that the networks are biconnected.

### A. Link Failure Recovery method (LFR)

In greedy routing, shortcut failures do not cause any problem in the routing because the spanning tree provides a distance-decreasing path between every two nodes. Therefore, the proposed recovery method is only applied in case of a tree-edge failure in the network. Protocols such as Bidirectional Forwarding Detection (BFD) [13] can be used for failure detection. Figure 2 illustrates the scenario of a shortcut failure in the network. In the right figure, the greedy path upon a shortcut failure is depicted by the dot line. As we see, the distance-decreasing path is still available via the tree. In order to provide a single tree-edge failure resiliency, we use two sets of coordinates for the network nodes. Two different spanning trees of the network for generation of these two sets of coordinates are required. As previously mentioned, we use colored trees which can be found for any biconnected network. Colored trees, also known as Blue and Red trees, are two trees rooted at the same node in the network with the following property: for every node in the network, there are two disjoint paths to the root node on these two trees. This does not imply that the two trees are completely edge-disjoint. However, an important property of them is that if an edge is common in both trees, the direction that you pass the edge to get closer to the root is opposite in the trees.

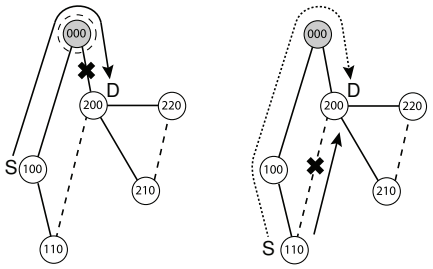


Fig. 2. The left figure depicts an example of local minimum in case of a tree-edge failure. On the right an example for a shortcut failure is illustrated.

There are several works in the literature to generate these types of trees [9], [10]. In [9], the authors propose to start by the root node and try to find a cycle which includes the root. Traversing the nodes of the cycle in one direction adds the path to the blue tree and traversing in the opposite direction determines the path for the red tree. The algorithm continues by finding the next cycle or path, starting from and ending to the visited nodes with unvisited intermediate nodes until all the network nodes are visited. Different ways of selecting the cycles/paths give different spanning trees with different properties.

Based on our previous studies, the degree and depth of a tree has effect on the quality of the greedy routing. This means that a spanning tree with higher degree and lower depth results into a greedy routing which the length of the paths are very close to the shortest path length between two nodes. Therefore, we try to generate such trees using the algorithm in [9]. The required steps are as follows:

1) Select the node with maximum degree as the root node.
2) Search for the shortest cycle which includes the root.
3) Determine the paths for the blue and red trees by traversing the cycle/path in two opposite directions.
4) Search for new paths/cycles (shortest), starting from a visited node closer to the root.
5) Go to step 3.

[10] proposes a distributed algorithm to generate the colored trees. They also use an ordering for the neighbors of the nodes to reduce the average path length on the trees.

Once the colored trees are generated for a network, the two sets of the coordinates can also be calculated based on the numbering in Section III-A. Every node knows the two coordinates for itself and also the two sets of coordinates for all of its neighbors. We refer to the two colored trees as primary and secondary trees. The greedy routing is always based on the primary tree unless otherwise mentioned. The proposed method for link failure recovery is referred to as *LFR*.

In order to recover a tree-edge failure, we need to distinguish the upward and downward failures. In the upward failure, the failing link is passed to get closer to the root of the tree while in the downward failure passing the failing link leads to a node deeper in the tree (further from the root).

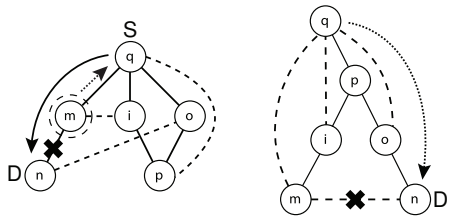


Fig. 3. Downward failure example. The primary and secondary trees are depicted. The primary greedy path is depicted by solid line in the left. The dot line represents the secondary path from failure detecting node (m) to the root and then from root to D.

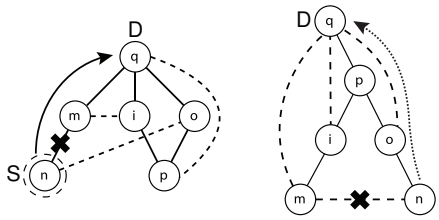


Fig. 4. Upward failure example. The primary and secondary trees are depicted. The primary greedy path is depicted by solid line in the left. The dot line represents the secondary path from failure detecting node (n) to the root (D).

Considering these two types of failure scenarios, the forwarding process upon reaching a downward failure is as follows:

- From the failure detecting node, greedy route to the root node based on the primary tree.
- From the root node greedy route to the destination on the secondary tree (disabling shortcuts).

Forwarding in the case of an upward failure is similar with the change of the trees:

- From the failure detecting node greedy route to the root node based on the secondary tree.
- From the root node greedy route to the destination on the primary tree (disabling shortcuts).

Figures 3 and 4 depict two examples for downward and upward failure scenarios. The colored trees are generated for the topology in Figure 1. In both figures, the primary tree is in the left and the secondary is in the right. In the left, the primary greedy path from S to D is depicted by solid line and the secondary path is partly based on the primary tree and partly based on the secondary tree. As the two paths from a node to the root are completely disjoint on the primary and secondary trees, greedy routing to the root based on one of the two trees (whichever that is still connected to the root) and switching to the other one in the root would definitely lead to the destination of the packet.

In LFR method, a packet is augmented with:

1) Mode field: This field determines three modes for the packets. i) normal greedy routing mode ii) recovery mode before root iii) recovery mode after root. The last two modes determine the recovery mode and also whether the packet has reached the root node or not.
2) Direction field: This field determines the direction of the failure (upward or downward).
3) Secondary Destination field: This field stores the destination coordinate based on the secondary tree.

A packet starts from the source node with the mode field set to *normal greedy routing*. Upon facing a failure in a node, the mode field is changed to the *recovery mode before root* and the direction field is set as well. Based on these two fields the packet is forwarded to the root node. In the root, the mode field is changed to *recovery mode after root*. Using these fields, each node knows how to forward the packet towards the intended destination.

The proposed method provides single failure resiliency using only two spanning trees. This makes the method significantly more memory efficient than the recovery methods in which the colored trees are calculated for every node in the network as the root.

Although the method recovers from every possible single tree-edge failure, it introduces extra load on the root node. In the following, we explain an improvement of the method to cope with this issue.

*1) Improved Link Failure Recovery method (ILFR):* In order to reduce the load on the root node, we need to introduce an extra field in the header of the packet. Using this field and an extra checking before forwarding a packet to the next hop we reduce the load on the root node significantly. The experimental results showed that on average for only 10% of the source-destination pairs which greedy routing between them faced the failing link, the packets would pass the root in the recovery mode. The extra field in the header of the packet is used to store the coordinate of the failure detecting node (primary/secondary coordinate for upward/downward failure). The general idea is that upon facing a failing link, we greedy route based on only one of the primary or secondary trees and if it is not possible to bypass the failing link, then we try to follow the previous method. In this way, not all the packets are routed to the root node.

In the upward failure, we route greedy based on the coordinates of the primary tree. In every node, we need to check if the next greedy hop is a node in the subtree below the failure detecting node. This is why the extra field in the header is required. Having this in mind, upon reaching an upward failure, the forwarding process in every node is as follows:

- If the current node is in the subtree below the failure detecting node:
  - Find the next greedy hop based on the primary tree (using shortcuts).
    - If the found next hop is not in the subtree below the failure detecting node, then go to that node.
    - Else, use secondary coordinates to go to the root.
- If the current node is out of the subtree below the failure detecting node:
  - Find the next greedy hop based on the primary tree.
    - If it is not in the subtree below the failure detecting node, then go to that node.

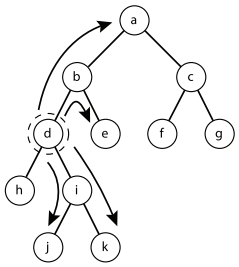


Fig. 5. The required backup paths for a node in the network.

- Else, find the next hop based on the primary tree (disabling shortcuts).

We observe that some extra checking is added in every node to see if the next hop is in the subtree below a certain node. This checking can be performed easily based on the coordinates of the nodes. As explained in Section III-A, the coordinates indicate the location of a node in the tree. Therefore, it is very easy to determine if a node is in a certain subtree (by just checking if they have common first fields in their coordinates).

The reason that this method leads to the destination is that, in a tree-edge failure scenario, the tree is disconnected to two subtrees, namely $A$ and $B$. In order to reach the destination, we need to reach to the subtree $B$ from subtree $A$. As the network is biconnected, there is always a shortcut to subtree $B$. If this shortcut is not found by greedy routing we need to use the path to the root (using secondary coordinate). Once we are in the correct subtree as destination, by checking that the shortcuts do not lead to subtree $A$, greedy routing will always lead to the destination.

The downward failure scenario is handled exactly the same however, the place of secondary and primary tree is changed. The reason that we can use the secondary tree and consider the failure as an upward failure is due to the properties of the colored trees. In case an edge is common in both trees, the direction to pass the edge to get closer to the root is opposite in the two trees. That is why we can consider a downward failure in the primary tree as an upward failure in the secondary tree. If the failing link is not common, this replacement does not cause any issue. However, as the secondary coordinates are used, we need to check that the destination (with secondary coordinate) is in the subtree below the failure detecting node or out of it. If it is out of it, then the procedure is exactly the same as upward failure but on the secondary coordinates. If not, normal greedy routing based on the secondary coordinates will lead to the destination.

### *B. Node Failure Recovery method (NFR)*

The similar method can be used to recover a node failure. The colored trees should be generated in such a way that the paths from every node to the root is node-disjoint on two trees. However, this method cannot recover the failure of the root node. Therefore, in this section, we propose a method which uses disjoint backup paths to recover any single node failure. In greedy routing based on the spanning tree of the network, the leaves of the tree do not require recovery.

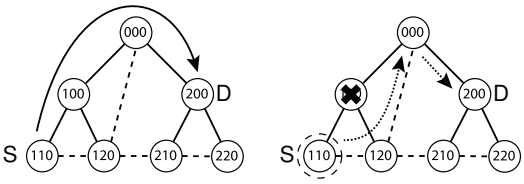


Fig. 6. Node failure example. The primary greedy path is depicted by solid line in the left. The dot line in the right, represents the secondary path.

The use of disjoint backup paths to recover failure is abundant in the literature. The general idea is to recover a node failure locally. This means that we use backup paths to bypass the failing node and then the greedy routing is resumed to the destination. As the proposed method is a protection scheme the backup paths are required to be determined before any failure in the network. We try to find a backup path between every two neighbors of a node in the tree. In other words, if we only consider the tree, for every node in the network, backup paths to its grandchildren, grandparent and siblings (nodes two hops away) are found. Figure 5 depicts the required backup paths for a node in the network.

Upon facing a failing node as the next hop, based on the coordinate of the destination, the node can decide which backup path to use. Greedy routing is resumed once the failing node is bypassed.

Storing an entry in the intermediate nodes along the backup paths implies a large memory overhead on those nodes. In order to avoid such memory overhead, we try to route greedy along the backup paths. This means that instead of storing an entry in every intermediate node, we check that up to which node along the path, greedy routing path is equal to the backup path. We call this node a hub node. Starting from this node, we continue checking to find the next hubs until we reach the last node of the backup path. Therefore, at the end of this checking we find some hub nodes which greedy routing from one to the next one is equal to the backup path. Now only these hub nodes are required to be stored in the protecting node. The forwarding process upon facing a node failure is as follows:

1) Select the correct backup path based on the destination coordinate (the one which leads to the direction of the destination, use tree numbering).
2) Add the hub nodes corresponding to that backup path to the header of the packet.
3) Greedy route to the first hub, second hub and so on until reaching the end node of the backup path.
4) Resume greedy routing to the destination.

Figure 6 depicts an example for node failure scenario. The correctness of the scheme can be proved by considering that the scheme actually follows the greedy route to the destination. Although, in case of facing a failing node a backup path to bypass the node which leads to the correct direction of the destination is used instead. In the following, we explain how the hub nodes can be determined using a few probe messages.

*1) Hub nodes determination:* We refer to a node as a protecting node once we are searching for backup paths to the nodes two hops away from that node. The hub nodes corresponding to the found backup paths are stored in this

node. These hubs are determined using probe messages. Once the backup paths are found, the protecting node starts the probing. Every node in the backup path knows the next hop. These entries can be removed once the hub nodes are found. Several types of probe messages are required. i) *Greedy-Check* message checks whether greedy routing to a node along the path gives the equal path as the backup path. ii) *Start-Check* message is used to inform a node along the path to start the same procedure for the rest of the nodes. The last two types are iii) *Ack* and iv) *NotAck* which will be described when to be sent.

- The protecting node sends a Greedy-Check message to the first node in the backup path.
- Upon arrival of the Greedy-Check message to the specified node in the destination field of the message, this node sends back an Ack (next hop included) to the protecting node.
- The protecting node stores the last node which sent back an Ack and sends another Greedy-Check to the next node along the path.

The above procedure continues up to the point that greedy routing path is not the same as the backup path.

- A node sends a NotAck to the protecting node if it detects that greedy routing to a node along the backup path is not the same as the backup path.
- Upon receiving the NotAck message, the protecting node determines the previously stored node as the first hub, and sends a Start-Check message to that hub node.
- The hub node starts sending Greedy-Check messages to the rest of the nodes along the path.
- If the hub node receives a NotAck, it sends back the last stored node to the protecting node.
- The protecting node repeats the same procedure with the new hub and continues to find all the hub nodes.

## V. Experimental Results

In this section, we evaluate the proposed methods through simulation. The simulation environment is mostly based on Python code. In order to evaluate the algorithms on large topologies, some parts of the code are optimized in C++. Libraries such as Networkx, Numpy and scipy are used for graph-based and numerical implementations.

In this experimental evaluation, we considered topologies with properties similar to the Internet. The topologies were evaluated at different scales from 100 up to 1000 nodes. They were generated based on the Barabasi-Albert (B-A) model [14]. In this model, a preferential attachment mechanism is used in order to generate random scale-free[1] topologies. Using this mechanism, the nodes with higher degree are more probable to receive new links which results into a power-law degree distribution with $\gamma$ equal to 3. The depicted results in every different experiment are averaged over 10 different random B-A topologies. These generated topologies have very similar properties in terms of degree distribution, number of links, etc. There are two sets of results, one for the LFR method and one for the NFR. The proposed schemes are evaluated for different aspects such as routing quality, memory overhead in network nodes, and overhead added to the packets. In order to have an overview of the performance of the schemes, we compare the results with gravity pressure method proposed in [7]. The reason to choose this method for comparison is that its greedy embedding and routing are also based on a tree and it is applicable in any type of topology.

### A. Stretch Evaluation

Using the proposed schemes, all the packets reach their intended destination without facing any dead end or loop. We evaluate the stretch for the proposed schemes. The stretch is defined as the ratio of the length of the path as produced by the greedy routing scheme, to the shortest path length for the same source-destination pair. In Figures 7 and 9, three different scenarios are compared. First the stretch without any failure in the network is calculated. Then the stretch in case of a single element failure is calculated both in the proposed schemes and in the gravity pressure method. The stretch in case of a failure is calculated as the ratio between the path length generated by greedy routing with the recovery method to the shortest path length after removing the faulty element from the topology. As the stretch results of the LFR and ILFR are very similar, we only report the results for the LFR method.

Figure 7 depicts the average stretch for link failure scenario in which the stretch over every possible pair of source-destination is calculated and averaged. The stretch is evaluated considering every tree-edge, one at a time as a failing edge and the average over all of the results is calculated. Each point in the graph represents the average value over 10 different B-A topologies. As we see, the proposed scheme has an insignificant effect on the stretch. The gravity pressure showed more negative effect on the routing quality however, the most performance gain in our proposed methods are in terms of overhead. Figure 8 depicts the stretch distribution in the networks with 1000 nodes. In this graph, the x axis shows the stretch values for all possible source-destination pairs in the network. As we see, almost 40% of the source-destination pairs have the stretch 1 and very few pairs have stretch higher than 2.5. The distributions before and after failure are very similar.

We performed the same experiment for node failure scenario. The network nodes were considered as failing nodes, one at a time, and the average stretch over all the cases was calculated. Figures 9 and 10 illustrate the stretch results for the NFR method. Again the effect on the stretch is very low in the proposed method and the gravity pressure results show more impact on the quality of the routing.

Based on the results of both link and node failure scenarios, the proposed schemes scale very well with the increase in the number of nodes. Observing this trend enables us to extrapolate the resulting performance up to large-scale topologies.

[1] A scale-free network is a network whose degree distribution follows a power law, at least asymptotically. That is, the fraction $P(k)$ of nodes in the network having $k$ connections to other nodes goes for large values of $k$ as $P(k) \approx c.k^{-\gamma}$. The value of $\gamma$ is typically in the range $2 < \gamma < 3$.

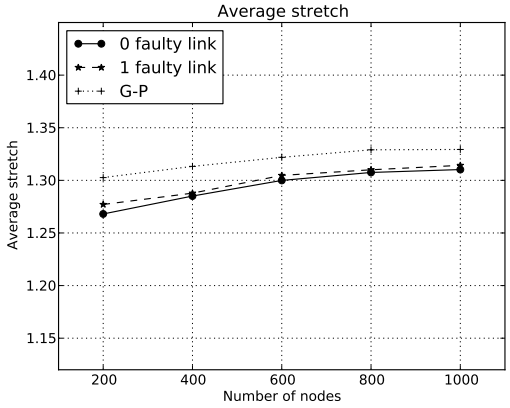


Fig. 7. The average stretch for scenarios without failure, single link failure in the LFR method and gravity pressure algorithm.

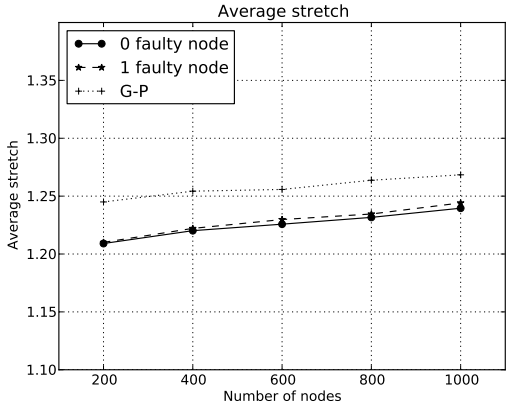


Fig. 9. The average stretch for scenarios without failure, single node failure in the NFR method and gravity pressure algorithm.

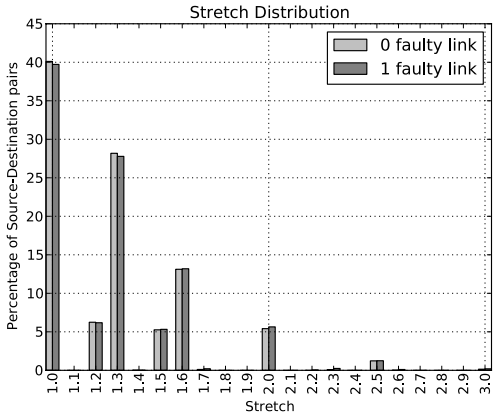


Fig. 8. Stretch distribution for scenarios without failure and single link failure in the LFR method. The network size is 1000.

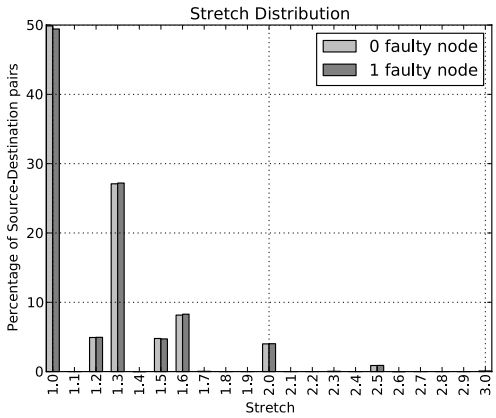


Fig. 10. Stretch distribution for scenarios without failure and single node failure in the NFR method. The network size is 1000.

### *B. Memory Overhead*

In this section, the implied overhead by the proposed schemes is evaluated. Figure 11 depicts the percentile of the required memory (number of entries) in the network nodes for the link failure scenario. As every node needs to store an extra coordinate for every neighbor (coordinate on the secondary tree), the memory overhead in every node is proportional to the degree of that node in the network. Therefore, the graph in Figure 11 somehow represents the degree distribution in the networks. Based on the results, in almost all the networks with different sizes, the third quartile of the values is around 10 and in the networks with 1000 nodes the maximum value is almost 120. In the scenario of a node failure, we need to calculate the number of hub nodes corresponding to the backup paths which are stored in every node. Figure 12 depicts the percentile of the required memory (number of entries) in the network nodes. The reason for the large differences between minimum and maximum values are due to the fact that the chosen spanning trees have a root node with maximum possible degree and lots of leaves (nodes with degree one). Therefore, for some nodes very few backup paths are required and for some so many backup paths are calculated. The values on top of the graph represent the maximum degree of the tree. In the networks of size 1000, the third quartile is almost 100 and the maximum value is 725.

In our next experiment, we evaluate the number of hub nodes in all the backup paths calculated in the network. Figure 13 illustrates the percentile of these numbers. As we see, in the networks of 1000 nodes, the maximum number of hubs in the backup paths reach up to $4$ nodes. This means that in the worst case the number of fields added to the header of a packet upon facing a failing node is $4$. As Gravity Pressure also uses a path trace in each packet, we calculated the size of the table stored in every packet which is in the Pressure mode of the algorithm in order to have a comparison with NFR. Here, we report the percentile of the size of the table in the packets in Pressure mode for networks with 1000 nodes and one single node failure. The $25$ percentile is $35$, the median is $82$, the $75$ percentile is $140$ and the maximum value is $223$. On average the size of this table is $87$. Comparing these results with the maximum number of hubs in NFR determines the efficiency of the proposed method in terms of overhead.

## VI. Discussions and Future Work

We have used colored trees (also known as independent trees) to guarantee a single link failure resiliency in the network. To recover more failures in the network, multiple

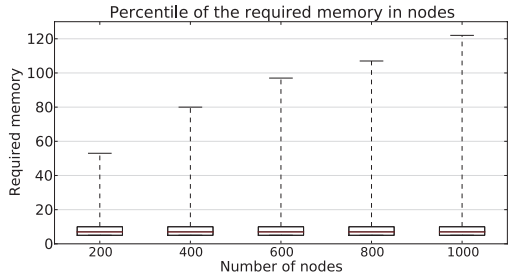


Fig. 11. Percentile of the required memory (nr. of entries) in network nodes in single link failure scenario.

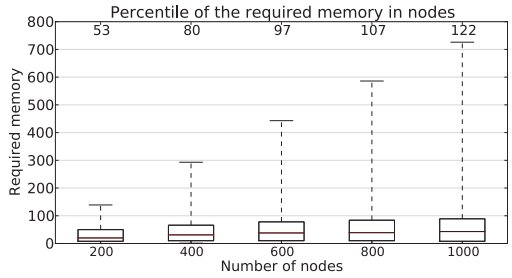


Fig. 12. Percentile of the required memory (nr. of entries) in network nodes in single node failure scenario.

independent trees are required. An interesting direction for future work is to recover multiple simultaneous failures using minimum number of independent trees which can be applied in large-scale networks. Another interesting work is to find a method to re-calculate the colored trees in such a way that the disruption is insignificant in order to recover multiple failures which are not simultaneous.

In the scenario of node failure, the proposed scheme can be used to recover multiple node failures. The only difference is to find multiple disjoint backup paths instead of only one. Using this method, the memory overhead in the nodes is also dependent on the number of failures that is guaranteed to be recovered. This scheme allows to recover multiple failures as long as the two failing nodes are not adjacent. So it is also a future plan to recover multiple node failures in any location.

## VII. Conclusion

We proposed two recovery methods for single link and node failure in greedy routing. We also proposed a simple embedding based on the spanning tree of the network in order to perform the greedy routing. In the scenario of a link failure, backup trees were used and for node failure scenarios, backup paths were considered. Using these techniques, there is no need to re-calculate the coordinates of the nodes. The proposed schemes are protection schemes which result into a fast switch-over. In the experimental evaluation, both methods showed interesting stretch characteristics compared to the existing alternatives and the added overhead to the packets was very low. The proposed methods can be used in large-scale networks due to their scalability, simplicity, low overhead and limited resource requirement.

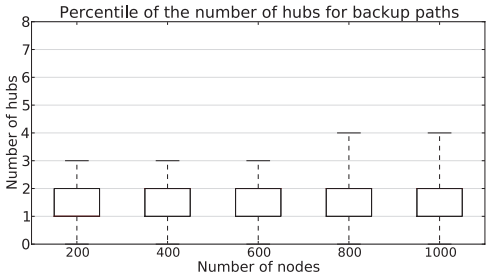


Fig. 13. Percentile of the number of hubs for backup paths in the network.

## Acknowledgment

This work is partly funded by the European Commission through the EULER project (Grant 258307), part of the Future Internet Research and Experimentation (FIRE) objective of the Seventh Framework Programme (FP7).

## References

[1] G. Huston, "BGP routing table reports," 2013, http://bgp.potaroo.net/.

[2] B. Karp and H. Kung, "GPSR: greedy perimeter stateless routing for wireless networks," in *Proceedings of the 6th annual international conference on Mobile computing and networking*. ACM, 2000, pp. 243–254.

[3] R. Fonseca, S. Ratnasamy, J. Zhao, C. Ee, D. Culler, S. Shenker, and I. Stoica, "Beacon vector routing: Scalable point-to-point routing in wireless sensornets," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*. USENIX Association, 2005, pp. 329–342.

[4] C. Papadimitriou and D. Ratajczak, "On a conjecture related to geometric routing," *Theoretical Computer Science*, vol. 344, no. 1, pp. 3–14, 2005.

[5] R. Kleinberg, "Geographic routing using hyperbolic space," in *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, 2007, pp. 1902–1909.

[6] D. Chen and P. Varshney, "A survey of void handling techniques for geographic routing in wireless networks," *IEEE Communications Surveys and Tutorials*, vol. 9, no. 1, pp. 50–67, 2007.

[7] A. Cvetkovski and M. Crovella, "Hyperbolic embedding and routing for dynamic graphs," in *INFOCOM 2009, IEEE*, 2009, pp. 1647–1655.

[8] S. Sahhaf, W. Tavernier, D. Colle, M. Pickavet, and P. Demeester, "Link failure recovery technique for greedy routing in the hyperbolic plane," 2012, http://dx.doi.org/10.1016/j.comcom.2012.08.023.

[9] M. Médard, S. Finn, and R. Barry, "Redundant trees for preplanned recovery in arbitrary vertex-redundant or edge-redundant graphs," *IEEE/ACM Transactions on Networking (TON)*, vol. 7, no. 5, pp. 641–652, 1999.

[10] S. Ramasubramanian, H. Krishnamoorthy, and M. Krunz, "Disjoint multipath routing using colored trees," *Computer Networks*, vol. 51, no. 8, pp. 2163–2180, 2007.

[11] A. Itai and M. Rodeh, "The multi-tree approach to reliability in distributed networks," *Information and Computation*, vol. 79, no. 1, pp. 43–59, 1988.

[12] J. Farkas, C. Antal, G. Tóth, and L. Westberg, "Distributed resilient architecture for ethernet networks," in *Design of Reliable Communication Networks, 2005.(DRCN 2005). Proceedings. 5th International Workshop on*. IEEE, 2005.

[13] D. Katz and D. Ward, "Bidirectional forwarding detection (BFD)," *RFC5880*, 2010.

[14] A. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–511, 1999.